electronics

Article

A Robust CNN for Malware Classification against Executable Adversarial Attack

Yunchun Zhang [1,*,†], Jiaqi Jiang [1,†], Chao Yi [1], Hai Li [1], Shaohui Min [1], Ruifeng Zuo [1], Zhenzhou An [2] and Yongtao Yu [2]

[1] School of Software, Yunnan University, Kunming 650095, China; jiangjiaqi@mail.ynu.edu.cn (J.J.); yichao@ynu.edu.cn (C.Y.); lihai@ynu.edu.cn (H.L.); minshaohui@ynu.edu.cn (S.M.); zhouruifa1999@gmail.com (R.Z.)
[2] Yunnan Key Laboratory of Smart City in Cyberspace Security, Yuxi Normal University, Yuxi 653100, China; an@yxnu.edu.cn (Z.A.); yyt@yxnu.edu.cn (Y.Y.)
* Correspondence: yczhang@ynu.edu.cn; Tel.: +86-871-6593-1540
† These authors contributed equally to this work.

Abstract: Deep-learning-based malware-detection models are threatened by adversarial attacks. This paper designs a robust and secure convolutional neural network (CNN) for malware classification. First, three CNNs with different pooling layers, including global average pooling (GAP), global max pooling (GMP), and spatial pyramid pooling (SPP), are proposed. Second, we designed an executable adversarial attack to construct adversarial malware by changing the meaningless and unimportant segments within the Portable Executable (PE) header file. Finally, to consolidate the GMP-based CNN, a header-aware loss algorithm based on the attention mechanism is proposed to defend the executive adversarial attack. The experiments showed that the GMP-based CNN achieved better performance in malware detection than other CNNs with around 98.61% accuracy. However, all CNNs were vulnerable to the *executable adversarial attack* and a fast gradient-based attack with a 46.34% and 34.65% accuracy decline on average, respectively. Meanwhile, the improved header-aware CNN achieved the best performance with an evasion ratio of less than 5.0%.

Keywords: malware detection; adversarial machine learning; malware images; Windows malware

Citation: Zhang, Y.; Jiang, J.; Yi, C.; Li, H.; Min, S.; Zuo, R.; An, Z.; Yu, Y. A Robust CNN for Malware Classification against Executable Adversarial Attack. *Electronics* **2024**, *13*, 989. https://doi.org/10.3390/electronics13050989

Academic Editor: Dimitris Kanellopoulos

Received: 13 January 2024
Revised: 22 February 2024
Accepted: 29 February 2024
Published: 5 March 2024

1. Introduction

The detection of malware is critical to many applications and networks as the number of malware incidents has increased dramatically due to the wide use of networks. It is necessary to design an automatic and intelligent malware-detection system to secure applications and users. With the rapid development of artificial intelligence, deep-learning-based malware classification has gained popularity due to its higher performance while avoiding laborious feature extraction [1,2]. Meanwhile, deep neural networks (DNNs) are vulnerable to adversarial attacks that apply minor perturbations to clean samples [3]. An adversarial attack tries to mislead the target deep learning models by creating input data (e.g., images, text) that have been slightly altered in a way that is often imperceptible to humans [3]. Adversarial attacks against malware-detection models try to modify the malware samples to mislead machine-learning-based detection systems while maintaining the malware's functionality. Thus, it is necessary to test whether existing deep-learning-based malware-classification models are robust against adversarial attacks.

It is still a challenging concern to generate effective adversarial malware [4,5]. In general, the attacker tries to generate adversarial samples with clean samples in the training data set. The target DNN produces a wrong prediction of the adversarial samples, either targeted or non-targeted. However, generating adversarial malware while preserving its original functionalities is still a nontrivial task. The authors in [6] generated adversarial malware by adding a new section in a precisely computed location within the original

file, usually at the end of the program and perturbed this section. Chen et al. [7] created adversarial malware in Windows by inserting non-functional codes in *Smali* codes. Meanwhile, most of the existing works assumed that the perturbations are applied on feature vectors while ignoring their mappings back to the original programs [8]. Furthermore, the research on adversarial malware contributes much to the development of a secure and robust malware-detection system. Aiming to design a robust neural network architecture for malware classification, this paper makes the following contributions:

- This study proposes a robust GMP-based CNN for detecting and classifying Windows malware. The designed model shares a similar network architecture with other neural networks except that the last pooling layer is global max pooling, which handles various sizes of malware images while preserving useful feature maps. Compared with other models, our model is characterized by its high efficiency and easy network configuration.
- This study designs an *executable adversarial attack* targeting the PE header files in the Windows operating systems while the maximum amount of distortions is predefined. The experiments showed that the *executable adversarial attack* gained a 64.7% attack success rate. The kernel size of around $[5, 7]$ is the most-appropriate, and a bigger kernel size for the first several convolutional layers is critical.
- This study further proposes a header-aware CNN to improve the robustness of the trained malware classifier. This model is an attention-mechanism-based model by adding a Euclidean-distance-based normalizer into its loss function. Through experiments, we show that the proposed header-aware CNN succeeds in decreasing the success rate from adversarial attackers to nearly 0, which means it is more robust than existing defensive mechanisms.

This paper is organized as follows: The background and main contributions are introduced in Section 1. We review the related works and recent advances for both robust machine learning and adversarial attacks in Section 2. Section 3 presents the structure of PE files, together with three deep-learning-based malware classification models. We show the implementation of the DeepFool-based adversarial attack targeting our trained classifier and the header-aware loss-based defensive strategy in Section 4. The experimental results and analyses are presented in Section 5. We conclude our work in the Section 6.

2. Related Works

2.1. Malware Classification and Detection

Based on whether program files are executed or not while collecting features, malware classification methods are grouped into two categories: dynamic and static classification [9].

Dynamic classification : By running program files, dynamic features are collected, including API call sequences, network communications, memory, and CPU states. Based on those dynamically extracted features, malware classification is performed by training different deep learning models. Athiwaratkun and Stokes [10] proposed a language model for API call sequence data using Long Short-Term Memory and the Gated Recurrent Unit. They also observed that a character-level CNN performed worse than their language model. Hou et al. [11,12] and Pektas et al. [13] recently converted API call sequences into a graph or a network topology.

Static classification: Signature matching, as an efficient method for malware classification, is incapable of detecting morphed or obfuscated malware. In contrast, machine-learning-based malware classification provides an effective way to detect or classify malware, especially when applied to malware gray-scale images. Nataraj et al. [14] first proposed a method for converting the malware binary program into a gray-scale image. By applying a feature extraction algorithm on the images, both the k-nearest neighbor (KNN) and CNN models were trained [15]. Other static features include bit sequences of the raw program [16] and code heterogeneity [17].

As dynamic classification needs a virtual machine (such as a sandbox) to collect dynamic features, some advanced attacks are capable of detecting their running in a virtual environment and, then, change their behaviors to evade detection.

2.2. Adversarial Attacks on DNNs

Szegedy et al. [18] demonstrated that conventional machine learning algorithms and neural networks are vulnerable to adversarial attacks. Many attack methods have been proposed under either white-box (attackers have accessibility to all models, parameters, classification results, and data sets) or black-box (where attackers know nothing about the models and data sets, except the output labels queried with an input sample) attacks. We have witnessed some well-known white-box attacks, including the fast gradient sign method [19], DeepFool [20], and the Jacobian-based Saliency Map Attack [21]. However, white-box attacks are hard to launch because attackers in real applications often have little information on the target models and training data samples. Some distinguished black-box attacks, including the momentum iterative fast gradient sign method [22] and zeroth-order optimization [23], are effective and highly transferable.

As far as malware is concerned, the adversarial malware should maintain its original malicious functionalities while misleading the target models. The mapping from computed perturbations to the original malware program varies greatly, as shown below:

- For API call sequences, feature addition is allowed while deletion is forbidden to maintain functionalities [24,25]. Chen et al. [7] proposed two strategies for mapping perturbations to the original files, including simple manipulation of the API and sophisticated manipulation of caller–callee relationships. In contrast, sophisticated manipulation changes non-functional code segments in *Smali* codes.
- For malware gray-scale images, both global and local perturbations are possible [26]. An adversarial attack on a few bytes at the end of each malware sample was designed and tested [6]. It only aims at achieving higher false negative rate (wrongly classify malware as benign). By analyzing malware images, an obfuscation-based adversarial-malware-generation method was proposed in [27].

Our proposed attacking method is closely related to [6], where perturbations are applied on a few specific bytes at the end of each program. Instead, we restricted our modification within the header part of a PE file by replacing specific bytes to avoid enlarging the file size.

2.3. Defense against Adversarial Attacks

Some defensive methods against both white-box and black-box attacks have been proposed, but a state-of-the-art solution is missing. Adversarial training [19], as one of the most-adopted defensive strategies, is robust to white-box attacks. Defensive distillation [28] made gradient calculation more difficult for attackers to generate adversarial samples. Recently, generative adversarial networks have proven to be the most-effective solution to defend against both white-box and black-box attacks [29].

As perturbations applied on malware gray-scale images are usually restricted within certain local sections, this paper aims to design a secure and robust defense to those attacks. The proposed attention-based defensive method targets an optimized loss function with a weighted normalizer computed based on the deviation between the perturbed image and the original image.

3. Methodology

As many malware programs target Windows systems, this paper aims at building a robust and secure deep learning model for malware detection and classification. A brief description of the structure of the PE format for Windows files is described first. Then, the procedure for generating a gray-scale image from the raw program is presented. Three CNNs with different pooling layers are designed and introduced for malware classification with gray-scale images as inputs.

3.1. PE Format

The PE format is an organized structure for executable files in Windows. The PE format contains the following three parts.

Header: The header part is used to store basic information, including the entry point, the file size, and a pointer to the next section. The header is further divided into four parts:

1. *DOS headers*, the beginning bytes of a PE executable, are essential and represent whether it is a basic DOS file.
2. *DOS stub* is used to show a message indicating whether the program can be run in DOS mode or not through the console.
3. *PE header or NT header*, the first byte of a PE executable representing whether it is a basic PE file or not.
4. *The optional header*, which contains descriptive information about a PE file, includes the entry point of the code, the number of sections, and the imported system APIs.

Tables: This section represents the characteristics of each file section and serves two major purposes: to map the file section into memory and to describe whether the file section is read-only or writable.

Sections: Both the data and code are stored in the sections. Even if different assemblers may generate these sections differently, the sections generally contain five parts:

1. *.text*, which contains the global and static variables.
2. *.data*, which contains the code instructions.
3. *.pdata*, which contains the debugging information.
4. .rsrc, which contains the resource information.
5. *.bss*, which contains the uninitialized variables.

The header part has contributed the most when classifying malware and has been a frequently attacked area by adversarial attacks. Therefore, this paper launches adversarial perturbation attacks on the header part.

Based on our observations of Windows PE files, this study proposes a robust deep learning model tailored to Windows malware classification. Illustrated in Figure 1, the system encompasses two sequential modules: the generation of an adversarial attack and the enhancement of malware classification robustness. The first module implements the *executable adversarial attack* devised in our research, generating adversarial malware images that can elude the classifiers effectively. Subsequently, the second module, known as robust malware classification, seeks to improve the model's robustness by incorporating both adversarial and clean samples during retraining. The operational principles of each module are described below.

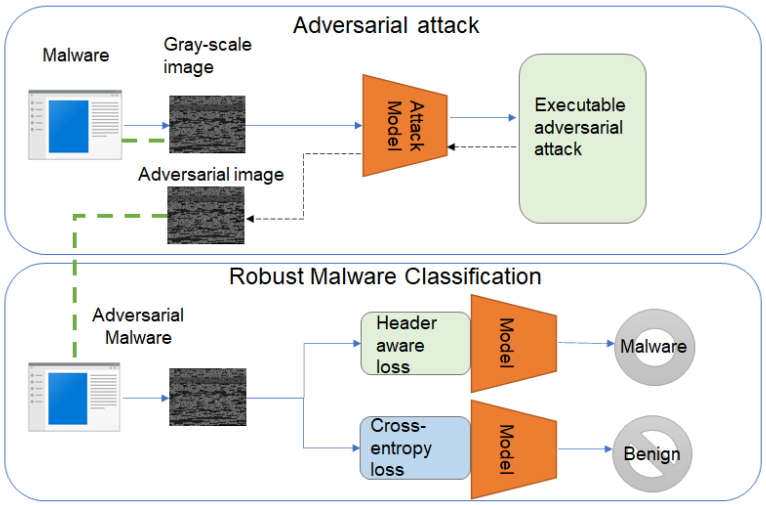

Figure 1. The overall system architecture for a robust malware classification CNN.

3.2. Malware Gray-Scale Image Construction

As deep-learning-based image-processing techniques gain momentum and achieve remarkable performance, it is common to convert malware programs into gray-scale images and, then, apply image-classification models. For each malware program, a gray-scale image is constructed by the following steps [14] as shown in Figure 2. First, each program is converted into a binary vector with every 8th bit representing an unsigned integer in the range of $[0, 255]$. Second, given the pre-defined image width (such as 128 bytes in this paper), each program is arranged in a row-by-row manner. As programs vary in size, their corresponding images also differ in length. Finally, we save the constructed gray-scale images in a lossless compression format, such as *.PNG*.

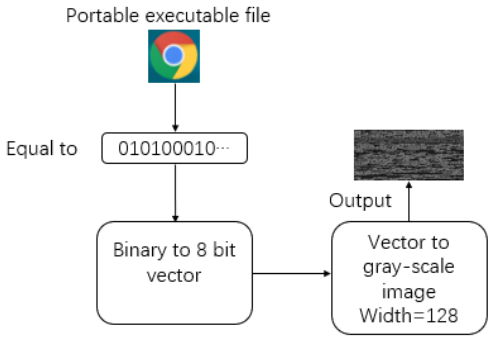

Figure 2. Major steps for constructing malware image from raw *.exe* file.

3.3. GMP-Based CNN for Malware Classification

When performing image classification, most neural networks take fixed-size images (e.g., 224×224) as the inputs. When inputting various sizes of images into neural networks, either the images are resized or the neural network architecture should be revised. However, changing the neural network architecture is more challenging than resizing the images. Consequently, various sizes of images are either cropped [30] or wrapped [31] into a fixed rectangular size (such as 224×224 or 128×128). It is shown that the performances under the cropping or warping methods are poor due to texture feature loss or global pattern distortion [32]. Therefore, proposing an end-to-end malware classifier by inputting malware images directly is urgently needed.

While the CNN is widely adopted for image classification, the conventional CNN only takes fixed-size images as the inputs. He et al. [32] proved that it is the last fully connected layer that requires fixed-size feature maps for classification. Therefore, this paper proposes three models for malware image classification with different pooling layers, GAP-based, GMP-based, and SPP-based CNNs, as shown in Figure 3.

3.3.1. GAP-Based and GMP-Based CNNs

Firstly, we designed a GAP-based and a GMP-based CNN, which both share the same network architecture, but with different pooling layers, as shown in Figure 3a. The last convolutional layer outputs 96 feature maps, which are averaged or maximized in the pooling layer. The averaged or maximized outputs are then forwarded to a fully connected layer for classification. The idea of applying GMP on the last pooling layer of a CNN to enable the CNN to take various sizes of images as the inputs is inspired by the "network in network" structure [33]. Compared with the GAP-based CNN, the GMP-based model is designed to convert the irregular image size into a fixed-length feature vector. As shown in Figure 3a, the last global max pooling layer takes feature maps derived from the former layer and outputs a $b \times 96 \times 1 \times 1$ fixed-length feature vector, where b means the batch size ($b = 4$ in this paper), which is adjustable under different running environments. In the GMP-based network, the cross-entropy is used as the loss function. Then, the malware

classification problem is modeled as an optimization problem minimizing the following objective function:

$$\underset{\theta}{Argmin}\, J(\theta) = \frac{1}{N}\sum_{i=1}^{N} -(y_i log(z_i) + (1-y_i) log(1-p_i)) \quad (1)$$

where N means the total number of output labels and y_i is a one-hot vector representing the label of the current malware sample with only one position set as 1. p_i is the probability that the sample belongs to the current malware family.

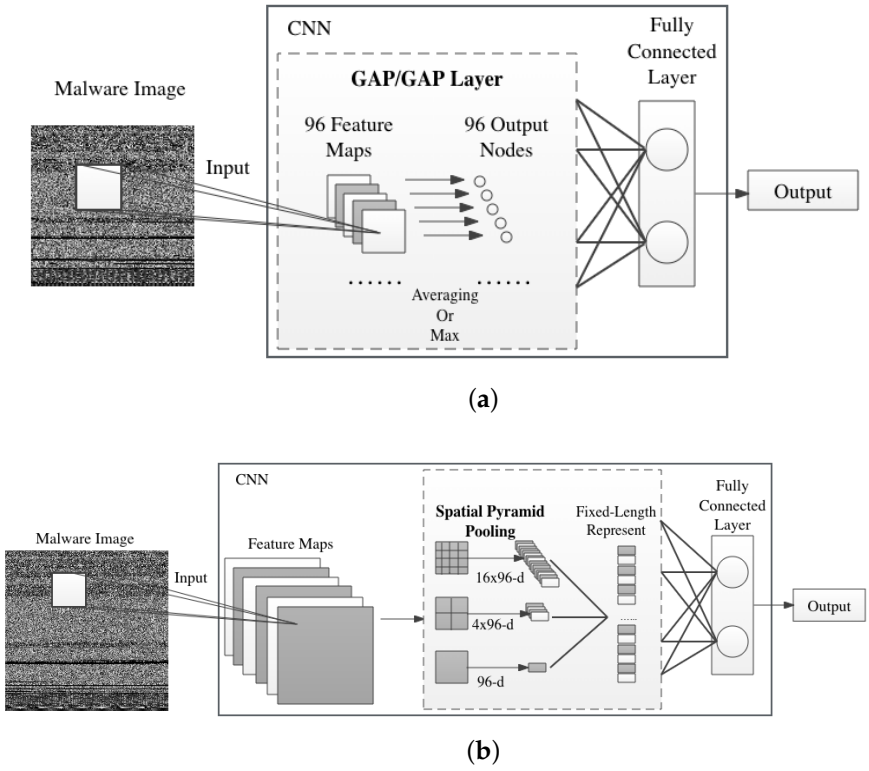

Figure 3. Network architecture of three models for malware classification. (**a**) Global average-/max-pooling-based CNN. (**b**) Spatial-pyramid-pooling-based CNN.

3.3.2. SPP-Based CNN

As shown in Figure 3b, the SPP-based CNN differs from the above two models in its spatial pyramid pooling layer. The parameters used in the SPP-based model are shown in Table 1. It is important to note that both the GAP-based model and GMP-based model share similar network parameter settings in Table 1, except for the last pooling layer. Our proposed SPP-based CNN is configured as follows:

- The neurons in the SPP-based network are in three dimensions: $width \times height \times depth$. All feature maps generated by each layer are also three-dimensional.
- Feature maps derived from the last convolutional layer are pooled N times (N is the number of pyramid layers and was set as $N = 3$). The parameters are optimized based on both the input image size and the position information where the current layer is located in the whole network architecture.
- Inspired by the design in [32], we used a 3-level pyramid with 4×4, 2×2, and 1×1 (in total, 21 bins) in our SPP-CNN.
- The outputs of the spatial pyramid pooling are $k \times M$-dimensional vectors, where $M = 21$ is the number of bins, while k is the number of filters in the last convolutional layer.

- The parameters used in Table 1 are computed and optimized by training the CNN 100 times when the network converges.

Table 1. The network architecture and parameters in SPP-based CNN.

Layers	Input Shape	Output Shape	Number of Output Features	Kernel Size
Convolutional layer 1	$302 \times 256 \times 1$	$151 \times 128 \times 6$	6	3×3
Convolutional layer 2	$151 \times 128 \times 6$	$76 \times 64 \times 12$	12	3×3
Convolutional layer 3	$76 \times 64 \times 12$	$76 \times 64 \times 24$	24	3×3
Convolutional layer 4	$76 \times 64 \times 24$	$76 \times 64 \times 48$	48	3×3
Convolutional layer 5	$76 \times 64 \times 48$	$38 \times 32 \times 64$	96	3×3
Spatial pyramid pooling	$38 \times 32 \times 64$	21×96	2016	Pyramid height = 3
Fully connected layer	21×96	2	2	NULL

4. Adversarial Attacks and Defensive Strategy

As deep learning models are trained and widely deployed in real applications for malware classification, the attacks targeting them are flourishing. As far as malware images are concerned, the adversarial attacks aim at generating a crafted image that is visually imperceptible to the human eye, but is wrongly classified by a deep learning model. While the majority of the existing adversarial attacks only work on malware gray-scale images, the mapping from those perturbed images to malware programs while maintaining their malicious functionalities is rarely considered. Therefore, how to bridge the gap between executable adversarial malware and the adversarial sample is worth researching.

Definition 1 (Adversarial sample [19])**.** *Given a sample x' generated by applying conventional adversarial attacks on the original clean sample x, an **adversarial sample** is defined as x', which can cause a victim machine learning model to produce an incorrect output (either targeted or non-targeted).*

Generating adversarial samples in the malware domain is a nontrivial task. Adversarial malware not only targets evading the classifiers, but also requires being executable after successful evasion. Therefore, we define the *executable adversarial malware* as follows.

Definition 2 (Executable adversarial malware)**.** *An **executable adversarial malware** is a malware that not only succeeds in evading the target detection models' detection or misleading the target model's with the wrong classification labels, but also preserves its malicious functionalities and/or file size.*

4.1. Synthesizing Executable Adversarial Malware

The DOS header of a Windows PE file is usually 120 bytes, while only 22 bytes are essential to determine whether the program is executable or not. By analyzing the PE file, we made the following observations:

- The contents within the addresses in the range of both $[2, 60]$ and $[78, 120]$, named as redundant header parts, are editable. Applying perturbations to them will not disrupt the program's run-time behavior.
- For malware classification models, the DOS header part is useless because they provide no malware-specific features for classification.

- The header part provides larger perturbation space (roughly about 10^{288} kinds of possible perturbation patterns are applicable) for attackers on the premise that there is no limitation on the distortion scale.

Based on the above observations, an *executable adversarial attack* is proposed. Inspired by DeepFool [20], the proposed *executable adversarial attack* works by applying DeepFool on the redundant header parts while perturbations are limited within a certain extent (such as ℓ_2 or ℓ_∞). The designed attack method on the Windows PE program is shown in Algorithm 1. The perturbed areas are restricted to the range of $[2, 60]$ and $[78, 120]$ in a PE header file. Therefore, both the semantic meaning and file size are preserved.

Algorithm 1 Executable DeepFool attack.

1: **input:** image x,
classifier f.
2: **output:** perturbation $\hat{r}$.
3: $region \leftarrow zero - vector - like(x)$,
4: $region[2:60, 78:120] \leftarrow 1$,
5: while $\text{sign}(f(x_i)) = \text{sign}(f(x_0))$
6: $\quad r_i \leftarrow r_i - \frac{f(x_i)}{\|\nabla f(x_i)\|_2^2} \nabla f(x_i)$,
7: $\quad r_i \leftarrow r_i * region$
8: $\quad x_{i+1} \leftarrow x_i + r_i$,
9: $\quad i \leftarrow i + 1$.
10: **return** $\hat{r} = \sum_i r_i$.

As shown in Algorithm 1, the algorithm firstly receives an image x and reads it into a vector *region*, as shown in line 3. Then, an iterative perturbation process is applied in a *while* statement. The perturbations added are computed based on the gradient, which leads to the fast approach to the nearest decision boundary by $\frac{f(x_i)}{\|\nabla f(x_i)\|_2^2} \nabla f(x_i)$. When perturbations are applied on the region outside $region[2:60, 78:120]$, they are masked by setting $r_i * region$. Then, the left perturbations are added to the former x_i. The whole process stops either when the attack succeeds, where $f(x_i) \neq f(x_0)$, or no more perturbations are applicable when $x_{i+1} = x_i$.

4.2. Robust Header-Aware CNN against Attack

As adversarial attacks have prevailed, a secure and robust malware classification model is in high demand. The defensive method proposed in this paper is based on the following observations:

1. To better defend against the PE header-oriented adversarial attacks, we redesigned the loss function and the optimized objective function in the GMP-based CNN by analyzing defensive distillation [34].
2. The perturbation area is critical. Because most malware images are expected not to have salient global features [35], existing adversarial malware is generated by applying perturbations locally instead of globally. Therefore, the proposed defensive mechanism should focus on those local regions.
3. By introducing the attention mechanism [36], our method makes a trade-off between classification accuracy and robust defensive capability. The size of the region indicated by the attention mechanism is compact and restricted in certain regions.

Based on the above observations and the loss function defined in Equation (1), the proposed models are optimized with a header-aware loss function as:

$$J(x,y;\theta) \leftarrow \frac{1}{N}\sum_{i=1}^{N} -(y_i log(z_i) + (1-y_i) log(1-p_i)) + \lambda \frac{1}{N}\sum_{i=1}^{N}\left(f^{'}(x_i;\theta) - f^{'}\left(x_i^{'};\theta\right)\right)^2 \quad (2)$$

where $f(\cdot)$ denotes the inference stage, $f^{'}(\cdot)$ is the feature extraction stage, θ is the weights of the model, and z_i represents the inference output. During each round, the distortions are computed and adjusted by a normalizer $\lambda \frac{1}{N}\sum_{i=1}^{N}\left(f^{'}(x_i;\theta) - f^{'}\left(x_i^{'};\theta\right)\right)^2$. This normalizer is computed by a weighted average derivation among two classification results between $f^{'}(x_i;\theta)$ and $f^{'}(x_i^{'};\theta)$, where x_i and $x_i^{'}$ represent the original image and the perturbed image, respectively.

The details of our defensive strategy are shown in Algorithm 2. In Algorithm 2, in line 7, the cross-entropy-based loss function is computed. Then, the added normalizer is computed based on the Euclidean distance between the original x_i and the perturbed $x_i^{'}$ by our executable adversarial attack, as shown in line 8. Based on the normalizer, the loss function is updated in line 10, while parameter θ is also updated in line 11 with a proper learning rate α on a newly updated gradient.

Algorithm 2 Header-aware CNN for robust malware classification.

1: **input:** images X,
labels Y,
classifier f which output the logit z and feature h,
learning rate α,
ignore rate λ.
2: **output:** weight of classifier θ.
3: for x,y in X,Y
4: $region \leftarrow one - vector - like(x)$,
5: $region[0:120] \leftarrow 0$,
6: $z, h \leftarrow f(x;\theta)$
7: $J(x,y;\theta) \leftarrow \frac{1}{N}\sum_{i=1}^{N}(1-y_i)log(1-p_i) - y_i log(z_i)$
8: $dist \leftarrow \lambda \frac{1}{N}\sum_{i=1}^{N}\left(f^{'}(x_i;\theta) - f^{'}\left(x_i^{'};\theta\right)\right)^2$
9: $z^{'}, h^{'} \leftarrow f(x \cdot region;\theta)$
10: $J(x,y;\theta) \leftarrow J(x,y;\theta) + dist$
11: $\theta \leftarrow \theta - \alpha \nabla_\theta J(x,y;\theta)$
12: **return** θ.

5. Experiments and Analyses

5.1. Data Sets and Experimental Environment Setup

First, both malware and benign programs were collected from Malekal (https://www.virustotal.com/gui/user/Malekal_morte (accessed on 20 January 2024)) and VirusTotal (https://www.virustotal.com/gui/home/upload (accessed on 20 January 2024)) to ensure that all samples were executable files. More than 2000 malware and 2000 benign programs were chosen for our experiments. We used 70% of the samples for training and the remaining 30% for validation. Second, all samples downloaded are standalone *.exe* programs. Third, all samples were preprocessed and converted into gray-scale images.

All models proposed were tested under the same environment. The running operating systems were configured with an Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40 GHz and GeForce RTX 2080TI (GPU) and run on the Ubuntu 16.04 operating system. The experiments are

performed by using the *Adam* [37] optimizer with a learning rate of 1×10^{-4} and with 30 epochs to train all models for malware classification.

5.2. Malware Classification

We designed two five-layer neural networks, including GAP-based CNN (GAP-CNN) and SPP-based CNN (SPP-CNN) [32], to test and compare their relative performance against our proposed GMP-based CNN. To be intuitive and persuasive, the three models were designed and evaluated in the same environment. Each model was trained by running the algorithm 10 times while using different combinations of training samples (known as the 10-fold cross-validation method). All performance metrics, including *accuracy*, *recall*, *precision*, *specificity*, F1, receiver operating characteristic curve (ROC), and area under ROC curve (AUC), were tested multiple times and averaged. The classification results are summarized in Table 2, where the values in boldface indicate the best value for each measure among the classifiers. Based on the results shown in Table 2, the following observations and analyses were made.

Table 2. Performance under three models.

Metrics	GMP-CNN	GAP-CNN	SPP-CNN
Accuracy	98.61%	93.75%	97.22%
Recall	97.65%	95.29%	90.59%
Precision	95.40%	87.10%	96.25%
Specificity	97.64%	90.90%	94.11%
F1	95.66%	92.08%	93.55%
AUC	0.99	0.95	0.95

First, it is clear that the GMP-based model outperformed the GAP-based and SPP-based ones on *accuracy*. It is shown that most classifiers, either conventional machine-learning-based or deep-learning-based, had an accuracy of [90.0–98.0%]. The reasons why the GMP-based CNN was superior to existing classifiers are three-fold:

- As conventional machine learning classifiers are seriously dependent on an isolated feature-extraction program, the features extracted are mostly numerical and suitable for gradient-based neural networks. However, those gradient-based networks are seriously attacked by adversarial attacks.
- As global pooling decreases the dimensionality of the feature maps and computational complexity, GAP requires that the number of feature maps in the last pooling layer equals the number of output classification labels. For new malware variants, the network architecture should be re-configured. Consequently, the GAP-based model is observed to have low scalability.
- GAP preserves more background information, while the GMP maintains more texture information. Thus, the GMP is more suitable for malware gray-scale images than GAP as malware images are characterized by their texture features with almost no distinctive objects (such as animal heads, human faces, etc.).

Second, besides *accuracy*, the GMP-based CNN also outperformed other networks on both *recall* and *specificity*. For malware detection, which is a typical binary classification problem, *recall* is more valuable than *specificity*. As shown in Table 2, the GMP-based CNN achieved the best *recall* and was 4.71% better on average than the other two classifiers. Unfortunately, the GMP-based model achieved a little less *precision* than the SPP-based one with 0.85%. However, a little sacrifice with respect to the false positive rate is worthwhile for malware classification because any false negative prediction may cause great damage to network security.

Third, integrated metrics such as the *F1* or AUC are better for performance comparison. Thus, we compared the *F1* under two networks, and it is shown that the GMP-based CNN was the best. If the number of malware and benign programs varied remarkably, it was necessary to test each classifier's performance under *specificity*. As shown in Table 2, the GMP-based CNN achieved better *specificity* than the SPP-based and GAP-based ones with 3.53% and 6.74% improvements, respectively.

Fourth, ROC under three CNNs is shown in Figure 4. As shown, GMP-based CNN achieves better performance on solving overfitting problem in malware detection because it shows the smoothest curve. Meanwhile, the curve under SPP-based CNN is smoother than that of GAP-based CNN.

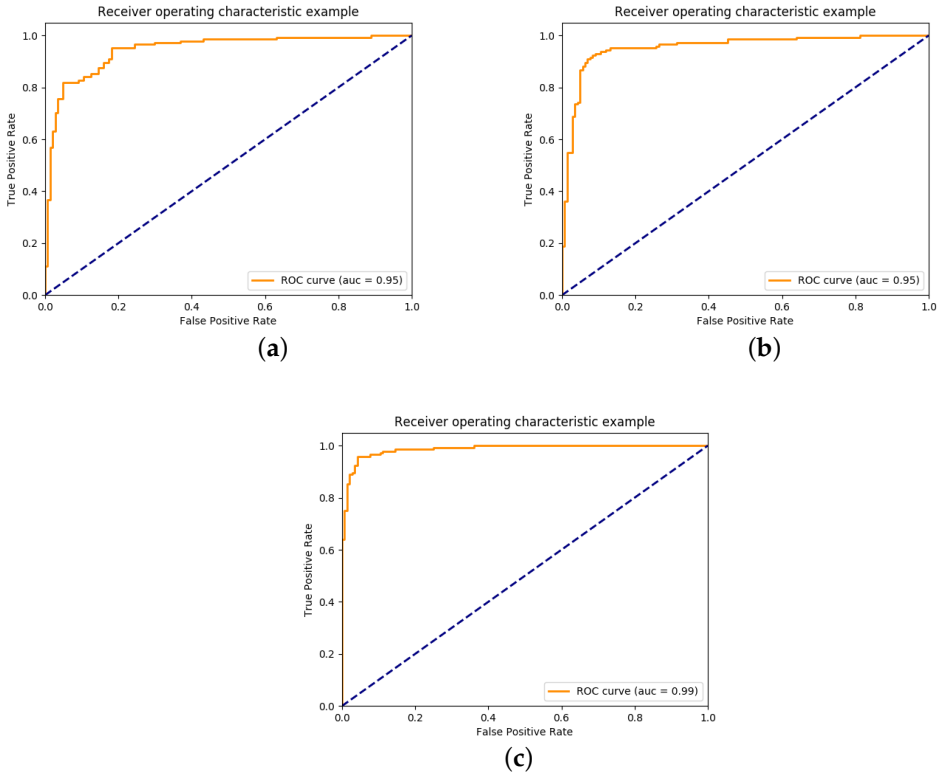

Figure 4. ROC under three CNNs for malware classification. (**a**) GAP-based model. (**b**) SPP-based model. (**c**) GMP-based model.

5.3. Performance under Different Adversarial Attacks

In this section, our proposed *executable adversarial attack* was compared with the attack proposed in [6]. The attack proposed in [6] is similar to the fast gradient method (FGM) [18], but the gradient is transformed by embedding weights. However, to maintain the program's original functionalities and semantic meanings, Kolosnjaji et al. [6] added a padded section at the end of the malware program for perturbation attacks. In contrast, this study improved the FGM and set the perturbed area to 128×128, while the iteration was set as 1000. To evaluate the effectiveness of each adversarial attack, the classification accuracy after running attacks against previously trained malware classification models is summarized in Table 3. We set the three five-layer neural networks with almost the same architecture, but different kernel sizes, where *Net*1, *Net*2, and *Net*3 had kernel sizes of $(3,3,3,3,3)$, $(5,3,3,3,3)$, and $(5,5,3,3,3)$, respectively.

The results in Table 3 show that the proposed *executable adversarial attack* is more effective than the FGM-based attack proposed in [6]. The detailed analyses and comparisons are summarized as follows:

- In general, *Net*3 showed the worst performance on the whole when compared with the other two networks with smaller kernel sizes. This means that a GMP-based model

with a bigger kernel size is more vulnerable when attacked by either an *executable adversarial attack* of an FGM-based attack.

- The attack in [6], with 12,800 bytes in total to apply perturbation, has more attacking space than our proposed *executable adversarial attack*. When attacked by the FGM-based method, the accuracy was stable at around 60.0–65.0%, which means that changing the kernel size had little effect on accuracy. While our attack has less attack space, it was more effective and efficient, with an 11.67% improvement on the accuracy decline on average compared to the FGM-based attack. However, the accuracy under our attack was influenced by the kernel size.
- FGM-based attack achieved the best results on recall, precision, and the F1 under *Net*3. When computing the degree of perturbations for an adversarial attack, the perturbed samples are supposed to approach the decision boundary for final classification. The whole attack process is highly sensitive to multiple factors, including the original samples, whether the gradients are computational or easy to compute based on the given samples due to gradient vanishing or explosion, etc. It is, thus, a common practice to compare different attacks under accuracy. All in all, the FGM-based attack performed better than our *executable adversarial attack* under some metrics. Thus, our attack won with the most-remarkable accuracy degradation.

While iterative attacks have already been observed to be more effective than single-step attacks, it is important to notice that the total number of distortions should be limited. As both the *executable adversarial attack* and FGM-based attack are applied on restricted areas of a malware image, the attacks should quickly converge. If more areas were allowed to be perturbed, the success rate would be improved, which means the accuracy under attack would drop accordingly. However, perturbations applied on other parts of a malware image have a higher probability to crack the original program. Therefore, we leave the perturbations applied on the whole image as our future works.

Table 3. Performance of the GMP-based CNN under two attacks, including executable adversarial attack and FGM-based attack.

Network	Executable Adversarial Attack (Ours)					FGM-Based Attack [6]				
	Accuracy	**Recall**	**Precision**	**Specificity**	**F1**	**Accuracy**	**Recall**	**Precision**	**Specificity**	**F1**
Net1	51.69%	22.44%	36.66%	72.46%	27.84%	65.00%	29.82%	65.38%	89.15%	40.96%
Net2	62.80%	42.55%	52.63%	75.67%	47.05%	60.74%	11.53%	66.66%	96.00%	19.67%
Net3	42.33%	64.91%	38.54%	26.25%	50.47%	66.14%	10.52%	30.76%	89.88%	15.68%

5.4. Performance Analyses of Robustness against Executable Adversarial Attack

5.4.1. Performance Analyses of Header-Aware CNNs

Adversarial training [19] is the benchmark technique for securing deep learning models against adversarial attacks. Adversarial training aims to extract highly representative features from adversarial samples. On the basis of the *executable adversarial attack*, it is essential for adversarial training to ignore the header part of the Windows PE file, which is frequently perturbed with a higher probability. Since the robustness of the malware classifier is a nontrivial task, this study further evaluated three models retrained with adversarial samples: adversarial training [19], Header-CNN2EAA (CNN retrained with adversarial samples generated by an *executable adversarial attack*), and Header-CNN2FGM (CNN retrained with adversarial samples generated by the FGM [18]). The ignore rate λ was set as 1.5×10^3 in the GMP-based CNN. As the file size increased, the ignore rate increased to 2×10^5 to defend against the attack proposed in [6]. The results are summarized in Table 4, where the values in the boldface indicate the best value for the evasion ratio among the three defensive mechanisms.

Table 4. Evasion ratio under different defensive models.

Network	Adversarial Training [19]	Header-CNN2EAA	Header-CNN2FGM
Net1	11.2%	2.1%	22.4%
Net2	14.5%	4.8%	9.8%
Net3	10.2%	0.0%	1.4%

As can be seen, Header-CNN2EAA was the best and achieved the lowest evasion ratio with different kernel sizes. We then made the following observations:

- Based on the results of the three adversarial-training-based models, adversarial training was effective at enhancing the robustness of all models. The reason is that these adversarial-training-based models succeed in extracting typical features of adversarial samples. Header-CNN2EAA was the best with the lowest evasion rate because of its ability to extract representative adversarial features applied by executable adversarial attacks.
- For both adversarial training and Header-CNN2EAA, the evasion ratio increased first and, then, dropped when increasing the kernel size. A smaller kernel size means fine granular attention to the perturbation areas. Therefore, Header-CNN2EAA achieved almost a 100% detection rate for Net3.
- While both adversarial training and the Header-CNN2FGM attack performed almost the same on average, the performance under Header-CNN2FGM varied sharply, while the performance under the adversarial training was stable.
- Header-CNN2EAA performed four-times better than the other models on average. Our header-aware CNN is also applicable to other malware collected under different operating systems, but should be improved based on detailed analyses to find which part of the code is redundant and, thus, suitable for attacks.

5.4.2. Performance under Different CNNs with Various Kernel Sizes

Given the image width (such as $width = 128$), the header section of a program may only have one or two rows of a gray-scale image. The texture feature patterns in those header parts are, thus, likely to be ignored by classification models. However, the feature maps can cover a larger perceptive field by increasing the kernel size. With a larger perceptive field, the kernel size was set in the range of $[3, 7]$ in each layer, as shown in Table 5. The first column represents the kernel size in each convolutional layer with five network layers in our designed models. If the kernel size was set to be 3 in each convolutional layer; this is represented as $(3, 3, 3, 3, 3)$. Constrained by the GPU memory, experiments with a kernel size greater than 9 are not included here, but the conclusions we derived should still hold.

As mentioned above in Algorithm 2, the kernel size determines the attention area when comparing the perturbed image with the original one. To evaluate how the kernel size impacts the performance of the models, the accuracy under the original GMP-based CNN and its accuracy under the *executable adversarial attack*, together with the evasion ratio under our header-aware defensive CNN, were compared.

First, we set a bigger kernel size for the former convolutional layers, as shown in Table 5. According to the results shown in the second column in Table 5, the classification accuracy increased by setting a larger kernel size for each layer. When increasing the kernel size, the accuracy fluctuated, but was always above 97.22% and was better than the state-of-the-art malware classifiers with the accuracy around 95.0%.

Table 5. Accuracy under both GMP-based CNN ($Accuracy_{GMP}$) and GMP-based CNN under executable adversarial attack ($Accuracy^{adv}_{GMP}$), together with evasion ratio under header-aware defensive model ($Evasion - Ratio_{def}$).

Kernel Size	$Accuracy_{GMP}$	$Accuracy^{adv}_{GMP}$	$Evasion - Ratio_{def}$
(3,3,3,3,3)	97.22%	51.69%	2.1%
(5,3,3,3,3)	98.61%	62.80%	10.5%
(5,5,3,3,3)	98.61%	42.33%	4.8%
(5,5,5,3,3)	97.22%	73.60%	0.0%
(7,5,5,5,5)	98.61%	62.20%	0.8%
(7,5,5,3,3)	97.92%	72.40%	0.07%

Second, each network was compared against the *executable adversarial attack* under the same settings. The accuracy under the *executable adversarial attack* is shown in the third column in Table 5. It is, thus, observed that, with a bigger kernel size, the accuracy against our *executable adversarial attack* fluctuated and stabilized around 70.0%. When compared with the accuracy of the GMP-based CNN with no attack, there was a 37.20% performance decline on average.

Third, the last column in Table 5 shows the defensive ability under our header-aware CNN against the *executable adversarial attack*. With a bigger kernel size, the evasion ratio dropped to nearly 0, which means that our defensive mechanism was effective. The evasion ratio converged and stabilized when kernel size was around $[5, 7]$ in the former three convolutional layers of the CNN.

5.5. Limitations and Secure Design Principles

Internal validity: To avoid the side-effect of intentional sample selection on experimental outcomes, this study collects a substantial number of Windows malware and constructs the used dataset through random sampling. Furthermore, this study solves the overfitting issue from two aspects. On the one hand, this study employs cross-validation techniques and utilizes regularization methods during the training process. On the other hand, the dropout layer in the CNN helps mitigate the overfitting. However, the designed methods apply to only Windows malware, and the design of the same method for Android malware is our future work.

Construct validity: We conducted experiments on a variety of model structures commonly used in malware-detection tasks to avoid structural biases that may arise from relying on a single model.

Conclusion validity: To ensure the statistical significance of the results, this study not only compares the accuracy of the models, but also conducts various statistical tests to determine the significance of the performance differences between the models.

Secure design principles: Since this study focuses on the feasibility of the *executable adversarial attack* and retraining a robust deep learning model against these attacks, the implementation of these models should comply with the application-level secure design principles [38]. Due to the limited effect, we leave this part for our future work.

6. Conclusions

Both black-box and white-box adversarial attacks are common in many machine-learning-based applications. Malware classification models are threatened within the context of the escalating arms race between hackers and security professionals. While existing adversarial attacks focus on the features used to train a classifier, it is critical to map those perturbations back into the original malware programs while preserving their functionalities. We designed a GMP-based CNN with malware gray-scale images as the inputs. This study then proposes the *executable adversarial attack*, which results in a sharp accuracy decline of the target classifiers. This attack is effective among different neural networks, while the perturbations computed are applied to the original program. A deep understanding of the adversarial attack is helpful when designing a robust and

secure machine learning model. Therefore, a secure and robust defensive-strategy-based attention mechanism is proposed against the *executable adversarial attack*, which applies to a PE file in Windows. Through detailed experiments, we showed that both the proposed *executable adversarial attack* and header-aware defense mechanisms are effective. However, it is important to note that our methods are specifically designed for Windows PE files. Given the widespread use of Android, iOS, and Linux/UNIX, developing a robust classifier for these operating systems becomes imperative. Since file formats differ across these systems from Windows, devising attacks tailored to the unique file formats of these operating systems remains a goal for our future research. The concepts and methodologies employed in this study lay the foundation for such future endeavors.

Furthermore, there is a pressing need for in-depth research to develop a state-of-the-art robust malware classifier. As the dynamics of adversarial attacks (from the attacker's perspective) and robust machine learning (from the security manager's standpoint) have both intensified recently, the conflict has escalated, becoming increasingly severe and challenging. Therefore, our future research will focus on designing a more-secure and -robust classifier against both white-box and black-box attacks. Instead of perturbing the header section of a PE file, generating adversarial malware while preserving their functions and semantic meanings is a possible direction.

Author Contributions: Conceptualization, Y.Z. and J.J.; methodology, Y.Z., J.J., C.Y. and H.L.; software, J.J., S.M. and R.Z., validation, J.J., S.M. and R.Z.; formal analysis, J.J., Y.Z. and Z.A.; investigation, Z.A. and Y.Y.; writing—original draft preparation, J.J. and Y.Z.; writing—review and editing, Y.Z. and J.J.; visualization, J.J.; supervision, Y.Z., C.Y. and H.L.; project administration, Y.Y.; funding acquisition, Z.A. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported in part by the Opening Foundation of Yunnan Key Laboratory of Smart City in Cyberspace Security (No. 202105AG070010) under Grants 202105AG070010-ZN-10 and 202105AG070010-JC-11.

Data Availability Statement: The data that support the findings of this study are available from the authors upon reasonable request.

Acknowledgments: The authors would like to thank the anonymous reviewers for their constructive comments and suggestions to improve the paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Chakraborty, T.; Pierazzi, F.; Subrahmanian, V.S. Ec2: Ensemble clustering and classification for predicting android malware families. *IEEE Trans. Dependable Secur. Comput.* **2020**, *17*, 262–277. [CrossRef]
2. Gibert, D.; Mateu, C.; Planes, J. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *J. Netw. Comput. Appl.* **2020**, *153*, 102526. [CrossRef]
3. Zhang, C.; Costa-Perez, X.; Patras, P. Adversarial attacks against deep-learning-based network intrusion detection systems and defense mechanisms. *IEEE/ACM Trans. Netw.* **2022**, *30*, 1294–1311. [CrossRef]
4. Li, X.; Li, Q. An IRL-based malware adversarial generation method to evade anti-malware engines. *Comput. Secur.* **2021**, *104*, 102118. [CrossRef]
5. Qiao, Y.; Zhang, W.; Tian, Z.; Yang, L.T.; Liu, Y.; Alazab, M. Adversarial malware sample generation method based on the prototype of deep learning detector. *Comput. Secur.* **2022**, *119*, 102762. [CrossRef]
6. Kolosnjaji, B.; Demontis, A.; Biggio, B.; Maiorca, D.; Giacinto, G.; Eckert, C.; Roli, F. Adversarial malware binaries: Evading deep learning for malware detection in executables. In Proceedings of the 2018 26th European Signal Processing Conference (EUSIPCO), Rome, Italy, 3–7 September 2018; pp. 533–537. [CrossRef]
7. Chen, X.; Li, C.; Wang, D.; Wen, S.; Zhang, J.; Nepal, S.; Xiang, Y.; Ren, K. Android HIV: A study of repackaging malware for evading machine-learning detection. *IEEE Trans. Inf. Forensics Secur.* **2020**, *15*, 987–1001. [CrossRef]
8. Li, F.Q.; Wang, S.L.; Liew, A.W.C.; Ding, W.; Liu, G.S. Large-scale malicious software classification with fuzzified features and boosted fuzzy random forest. *IEEE Trans. Fuzzy Syst.* **2021**, *29*, 3205–3218. [CrossRef]
9. Faruki, P.; Bharmal, A.; Laxmi, V.; Ganmoor, V.; Gaur, M.S.; Conti, M.; Rajarajan, M. Android security: A survey of issues, malware penetration, and defenses. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 998–1022. [CrossRef]

10. Athiwaratkun, B.; Stokes, J.W. Malware classification with LSTM and GRU language models and a character-level CNN. In Proceedings of the 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), New Orleans, LA, USA, 5–9 March 2017; pp. 2482–2486. [CrossRef]
11. Hou, S.; Saas, A.; Chen, L.; Ye, Y. Deep4maldroid: A deep learning framework for android malware detection based on linux kernel system call graphs. In Proceedings of the 2016 IEEE/WIC/ACM International Conference on Web Intelligence Workshops (WIW), Omaha, NE, USA, 13–16 October 2016; pp. 104–111. [CrossRef]
12. Hou, S.; Ye, Y.; Song, Y.; Abdulhayoglu, M. Hindroid: An intelligent android malware detection system based on structured heterogeneous information network. In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 13–17 August 2017; pp. 1507–1515. [CrossRef]
13. Pektaş, A.; Acarman, T. Deep learning for effective Android malware detection using API call graph embeddings. *Soft Comput.* **2020**, *24*, 1027–1043. [CrossRef]
14. Nataraj, L.; Karthikeyan, S.; Jacob, G.; Manjunath, B.S. Malware images: Visualization and automatic classification. In Proceedings of the 8th International Symposium on Visualization for Cyber Security, New York, NY, USA, 20 July 2011; pp. 1–7. [CrossRef]
15. Gibert, D.; Mateu, C.; Planes, J.; Vicens, R. Using convolutional neural networks for classification of malware represented as images. *J. Comput. Virol. Hacking Tech.* **2019**, *15*, 15–28. [CrossRef]
16. Zhang, Y.; Sui, Y.; Pan, S.; Zheng, Z.; Ning, B.; Tsang, I.; Zhou, W. Familial clustering for weakly-labeled android malware using hybrid representation learning. *IEEE Trans. Inf. Forensics Secur.* **2020**, *15*, 3401–3414. [CrossRef]
17. Tian, K.; Yao, D.; Ryder, B.G.; Tan, G.; Peng, G. Detection of repackaged android malware with code-heterogeneity features. *IEEE Trans. Dependable Secur. Comput.* **2020**, *17*, 64–77. [CrossRef]
18. Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; Fergus, R. Intriguing properties of neural networks. In Proceedings of the 2nd International Conference on Learning Representations, Banff, AB, Canada, 14–16 April 2014.
19. Goodfellow, I.J.; Shlens, J.; Szegedy, C. Explaining and harnessing adversarial examples. In Proceedings of the 3rd International Conference on Learning Representations, San Diego, CA, USA, 7–9 May 2015.
20. Moosavi-Dezfooli, S.M.; Fawzi, A.; Frossard, P. Deepfool: A simple and accurate method to fool deep neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 1 July 2016; pp. 2574–2582. [CrossRef]
21. Papernot, N.; McDaniel, P.; Jha, S.; Fredrikson, M.; Celik, Z.B.; Swami, A. The limitations of deep learning in adversarial settings. In Proceedings of the 2016 IEEE European Symposium on Security and Privacy (EuroS&P), Saarbruecken, Germany, 21–24 March 2016; pp. 372–387. [CrossRef]
22. Dong, Y.; Liao, F.; Pang, T.; Su, H.; Zhu, J.; Hu, X.; Li, J. Boosting adversarial attacks with momentum. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 9185–9193. [CrossRef]
23. Chen, P.Y.; Zhang, H.; Sharma, Y.; Yi, J.; Hsieh, C.J. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, New York, NY, USA, 3 November 2017; pp. 15–26. [CrossRef]
24. Grosse, K.; Papernot, N.; Manoharan, P.; Backes, M.; McDaniel, P. Adversarial perturbations against deep neural networks for malware classification. *arXiv* **2016**, arXiv:1606.04435.
25. Al-Dujaili, A.; Huang, A.; Hemberg, E.; O'Reilly, U.M. Adversarial deep learning for robust detection of binary encoded malware. In Proceedings of the 2018 IEEE Security and Privacy Workshops (SPW), San Francisco, CA, USA, 24 May 2018; pp. 76–82. [CrossRef]
26. Su, J.; Vargas, D.V.; Sakurai, K. One pixel attack for fooling deep neural networks. *IEEE Trans. Evol. Comput.* **2019**, *23*, 828–841. [CrossRef]
27. Park, D.; Khan, H.; Yener, B. Generation & evaluation of adversarial examples for malware obfuscation. In Proceedings of the 2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA), Boca Raton, FL, USA, 16–19 December 2019; pp. 1283–1290. [CrossRef]
28. Papernot, N.; McDaniel, P.; Wu, X.; Jha, S.; Swami, A. Distillation as a defense to adversarial perturbations against deep neural networks. In Proceedings of the 2016 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 23–25 May 2016; pp. 582–597. [CrossRef]
29. Jin, G.; Shen, S.; Zhang, D.; Dai, F.; Zhang, Y. Ape-gan: Adversarial perturbation elimination with gan. In Proceedings of the ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brighton, UK, 12–17 May 2019; pp. 3842–3846. [CrossRef]
30. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. In Proceedings of the 25th International Conference on Neural Information Processing Systems—Volume 1, Red Hook, NY, USA, 3–8 December 2012; pp. 1097–1105.
31. Donahue, J.; Jia, Y.; Vinyals, O.; Hoffman, J.; Zhang, N.; Tzeng, E.; Darrell, T. Decaf: A deep convolutional activation feature for generic visual recognition. In Proceedings of the 31st International Conference on Machine Learning, Bejing, China, 22–24 January 2014; pp. 647–655.
32. He, K.; Zhang, X.; Ren, S.; Sun, J. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* **2015**, *37*, 1904–1916. [CrossRef] [PubMed]
33. Lin, M.; Chen, Q.; Yan, S. Network in network. *arXiv* **2016**, arXiv:1312.4400.

34. Carlini, N.; Wagner, D. Defensive distillation is not robust to adversarial examples. *arXiv* **2016**, arXiv:1607.04311.
35. Yakura, H.; Shinozaki, S.; Nishimura, R.; Oyama, Y.; Sakuma, J. Malware analysis of imaged binary samples by convolutional neural network with attention mechanism. In Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, New York, NY, USA, 19–21 March 2018; pp. 127–134. [CrossRef]
36. Xu, K.; Ba, J.; Kiros, R.; Cho, K.; Courville, A.; Salakhudinov, R.; Zemel, R.; Bengio, Y. Show, attend and tell: Neural image caption generation with visual attention. In Proceedings of the 32nd International Conference on Machine Learning, Lille, France, 7–9 June 2015; pp. 2048–2057.
37. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. *arXiv* **2014**, arXiv:1412.6980.
38. Ebad, S.A. Exploring How to Apply Secure Software Design Principles. *IEEE Access* **2022**, *10*, 128983. [CrossRef]